Practica 3 TSI

José Manuel Pérez Lendinez

June 9, 2019

Contents

1 Ejercicio 1

El siguiente problema a resolver (problema-zeno-V01.pddl) consiste en transportar 3 personas (inicialmente en las ciudades C1, C2 y C3) a la ciudad C5, considerando que el avión está en la ciudad C4. Se asume al igual que en el problema ejemplo que no hay restricciones de fuel.

El problema en este ejercicio viene dado porque no teníamos en cuenta que el avión y el pasajero no estén en la misma ciudad. Esto lo solucionamos añadiendo un nuevo método que llamara a un avión que esta situado en otra ciudad.

Cuando se cumpla este caso se llamara a el avión a la ciudad donde se encuentra el pasajero, se hará el board y se mandara el avión a la cidudad destino

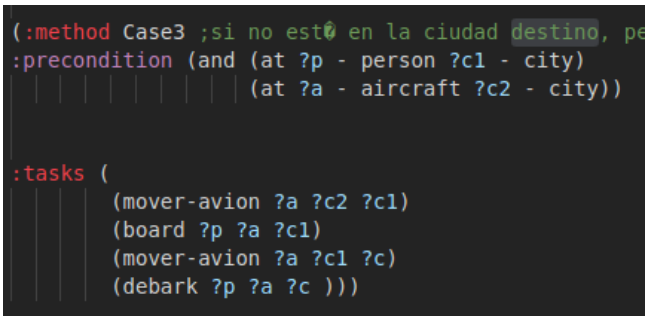

Figure 1

2 Ejercicio 2

En este caso tendremos que añadir las opciones necesarias para tener en cuenta el fuel de un avión. En este caso se tendrá que tener en cuenta que para ir de una ciudad a otra necesitaremos tener le fuel necesarios. Para esto crearemos un derived que nos devuelva el verdadero cuando tengamos fuel suficiente para ir de una ciudad a otra.

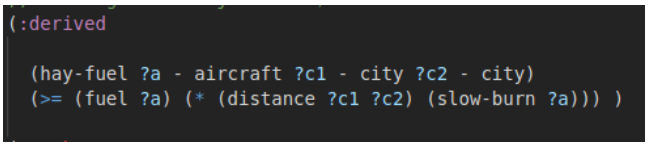

Figure 2

Con esto ya podemos modificar el task mover avión utilizando esta función. Se tendrá dos method uno cuando tengamos fuel que simplemente realizara el fly y otro en el caso de no tener fuel donde se realizara un refuel antes del fly.

```
(:method fuel-suficiente
 :precondition (hay-fuel ?a ?c1 ?c2)
 :tasks (
          (fly ?a ?c1 ?c2)
         )
 )
 (:method fuel-no-suficiente
 :precondition (not(hay-fuel ?a ?c1 ?c2))
 :tasks (
          (refuel ?a ?c1)
          (fly ?a ?c1 ?c2)
         )
 )
```

Figure 3

3 Ejercicio 3

En este caso se tendrá que diferenciar entre le vuelo rápido y lento. Dándole prioridad a los vuelos rápidos. Se tiene que añadir un nuevo derived igual que en el ejercicio anterior, con la diferencia de que tendrá en cuenta cuanto gasta un vuelo rápido en vez de uno lento.

```
(:derived

  (hay-fuel-rapido ?a - aircraft ?c1 - city ?c2 - city)
  (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a))) )
```

Figure 4

El derived del ejercicio anterior le cambiaremos el nombre ha hay-fuel-rápido.

También tendremos que tener dos nuevos derived para comprobar si sobrepasamos el limite máximo de fuel que se definirá para este ejercicio. Uno de ellos comprobara si lo sobrepasamos volando rápido y el otro volando lento.

```
(:derived

  (sobrepasa-limite-lento ?a - aircraft ?c1 - city ?c2 - city)
   (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (slow-burn ?a)) ) ))

(:derived

  (sobrepasa-limite-rapido ?a - aircraft ?c1 - city ?c2 - city)
   (> (fuel-limit)(+ (total-fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a)) ) ))
```

Figure 5

Una vez tengamos las funciones preparadas podremos modificar en método mover-avion para darle prioridad a los vuelos radios. Para esto podremos

primero los task que realizan el vuelo rápido y los últimos los task que realizan los vuelos lentos. Primero iran los method de vuelo rapido. Donde se usara las funciones que tendra en cuenta que no sobrepasa el limite y que tenemos fuel al volar rapido.

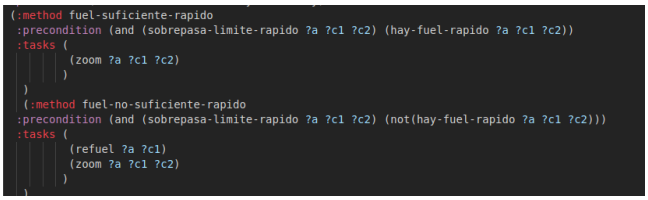

Figure 6

Después de estos dos se tendrán los dos de vuelo lento para ello las funciones de vuelo lento.

4 Ejercicio 4

En este caso tendremos un limite de pasajeros por avión que lo representaremos con un predicado (= (limite-pasajeros a1) 10) y otro para representar el numero de pasajeros ya montados que sera (= (personas-montadas a1) 0). A1 representaría a ana avión. Se cambiara las primitivas board y debark para tener en cuenta el limite de pasajeros. En board añadiremos una nueva restricción para que sola pueda aumentar pasajeros cuando no se supere limite y incremente el predicado personas-montadas. En el caso de debark se tendría que diminuir el predicado, no miraremos si estamos por debajo de 0 porque para llamar a un debark el pasajero tendrá que estar en el avión y por tanto anteriormente se ha dado un board.

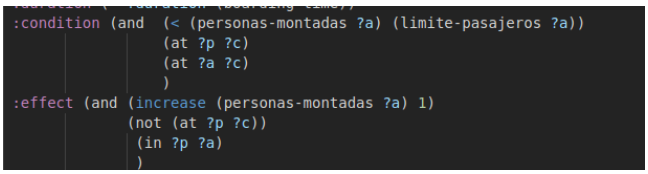

Figure 7: Board

```
:condition (and (> (personas-montadas ?a) 0)
                (in ?p ?a)
                (at ?a ?c)
                )
:effect (and  (not (in ?p ?a))
              (at ?p ?c)
              (decrease (personas-montadas ?a) 1)))
```

Figure 8: Debark

Para poder embarcar varios pasajeros en una misma avión vamos a realizar una función recursiva que sera la encargada de realizar todo lo relacionado con embarcar, desembarcar y movimiento de aviones. Esta función tendrá un total de 5 method. Seran los siguientes:

1. Montado en avión y avión en ciudad destino: En este caso el avión ya se encuentra en la ciudad de destino y el pasajero esta montado en le avios. Se realizara un debark en esta ciudad.

```
(:method montado-avion-en-destion
   :precondition (and (in ?persona - person ?avion - aircraft)
                      (at ?avion - aircraft ?origen - city)
                      (destino ?persona - person ?destino - city)
                      (= ?origen ?destino))
   :tasks (
     (debark ?persona ?avion ?destino)
     (planificador)
   )
 )
```

Figure 9

2. No montado en avión y avión en la ciudad: En este caso sera en el que los pasajeros hagan el board para subir a el avión. Se mirara que la ciudad no sea destino y que el pasajero y el avión estén en la misma ciudad.

```
(:method sin-avion-y-avion-en-ciudad
  :precondition (and (at ?persona - person ?c1 - city)
                     (at ?avion - aircraft ?c2 - city)
                     (hay-espacio ?avion)
                     (destino ?persona - person ?destino - city)
                     (= ?c1 ?c2)
                     (not(= ?c1 ?destino))
                     )
  :tasks (
    (board ?persona ?avion ?c1)
    (planificador)
  )
)
```

Figure 10

3. No montado en avión y sin avión en esa ciudad: En este caso se llamara a

un avión para que llegue a la ciudad en la que esta el pasajero sin montar en ninguna avión y cuando esa ciudad no sea su destino.

```
(:method sin-avion-y-avion-no-ciudad
  :precondition (and (at ?persona - person ?c1 - city)
                     (at ?avion - aircraft ?c2 - city)
                     (destino ?persona - person ?destino - city)
                     (not(= ?c1 ?c2))
                     (not(= ?c1 ?destino)))
  :tasks (
    (mover-avion ?avion ?c2 ?c1)
    (planificador)
  )
)
```

Figure 11

4. Montado en avión y avión no esta en destino: En este caso el pasajero estará montado en un avión y esta avión estará en una ciudad que no concuerda con la ciudad destino del pasajero. Se realizara un vuelo hasta la ciudad de destino del pasajero.

```
(:method montado-avion-no-destion
  :precondition (and (in ?persona - person ?avion - aircraft)
                     (at ?avion - aircraft ?origen - city)
                     (destino ?personas - person ?destino - city)
                     (not (= ?origen ?destino)))
  :tasks (
    (mover-avion ?avion ?origen ?destino)
    (planificador)
  )

)
```

Figure 12

5. En Destino: Es el caso base en el que no se volverá a realizar la recursividad. Se da cuando el pasajero esta en destino y no esta subido en un avión.

```
(:method en-destino
  :precondition (and (at ?p - person ?origen - city)
                     (destino ?p - person ?destino - city)
                     (= ?origen ?destino))
  :tasks (

  )
```

Figure 13

No he realizado la parte de que cada avion tenga un tiempo limite.